

EAAT User Manual

A guide how to use *Class Modeler & Object Modeler*

Dept. of Industrial Information and Control Systems, KTH

August 2013

Content

EAAT User manual	3
1 What is the EA tool? - Background information	4
2 The Predictive, Probabilistic Architecture Framework	5
3 The Class modeler	6
4 The Object modeler	15
4.1 General info	15
4.2 Modeling process	15
4.2.1 Model structure	16
4.2.2 Evidence.....	17
4.2.3 Calculation	19
4.2.4 Exporting	21
4.3 Sampling methods	22
4.3.1 Forward sampling	23
4.3.2 Forward sampling with evidence injection.....	24
4.3.3 Rejection sampling	24
4.3.4 Metropolis- Hastings sampling	25
4.4 Problems	26
7. Specification of analysis	28
4.5 Mathematical functions	28
4.5.1 Probability Distribution Functions	28
4.5.2 Cumulative Distribution Functions	29
4.5.3 Nested distributions	30
4.6 OCL	31
5 FAQ	33
5.1.1 Class Modeler	33
5.1.2 Object Modeler	33
6 References	34

EAAT User manual

This document is a user manual for the Enterprise architecture Analysis tool (EAAT). For more information about the tool project, please consider <http://www.ics.kth.se/eaat>.

1 What is the EA tool? - Background information

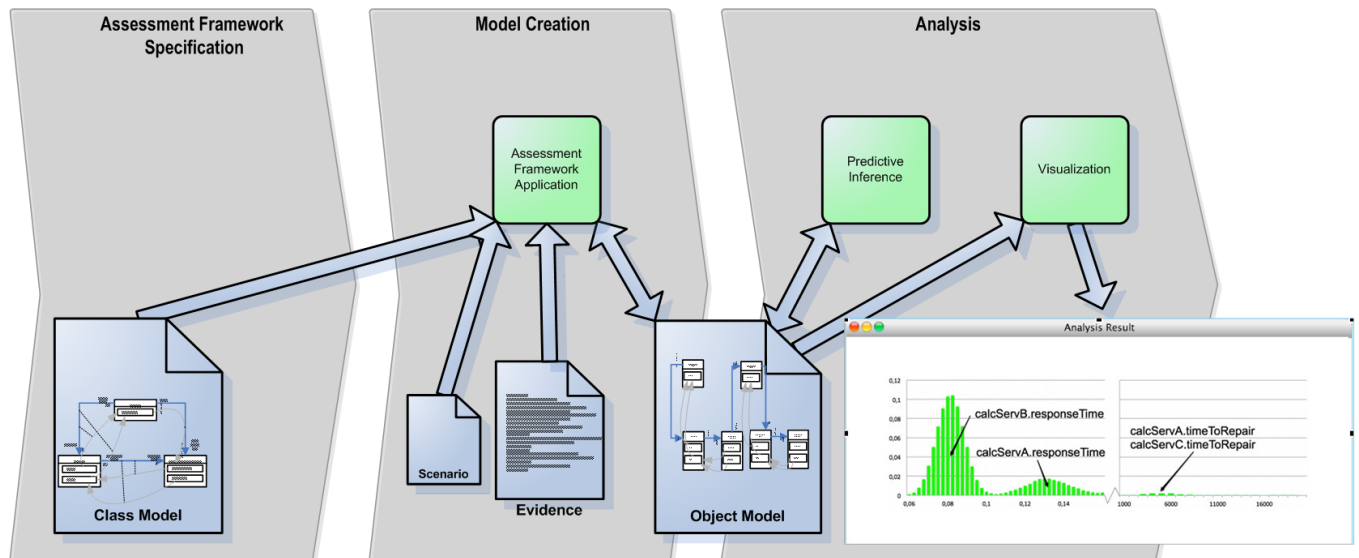


Figure 1 The supported enterprise architecture analysis method

Several methods exist for performing analysis of enterprise architecture models for decision support. The tool described in this manual supports the method presented in (P. Johnson & Ekstedt, 2007). This method is depicted in Figure 1. First, an extended metamodel is created. Following the UML nomenclature, the authors refer to it as class model. This class model has an extended meaning. It describes not only the allowed content of the models, but also how characteristics of the model impact each other with regard to chosen criteria for architecture analysis. For those characteristics of the model general data describing them are included too. In the second step, scenarios of interest are identified. Each scenario is described as an object model instantiating the previously created class model. For a particular scenario one typically wants to replace specify attribute values, for some or all attributes that are part of the model. This is done in order to provide a more specific description. In the nomenclature of probabilistic inference, such instance-specific data is called evidence. In the final step, analysis, quantitative values of the models' quality attributes are inferred and the results are visualized.

The following section (Section 2) explains the underlying theory of the analysis framework used in the tool. Following the method visualized in Figure 1 the tool consists of two components. The usage of the first component, the class modeler, is the content of Section 3. Section 4 follows and explains the usage of the object modeler. For additional information on the theory and architecture of the tool please consider <http://www.ics.kth.se/eaat>.

2 The Predictive, Probabilistic Architecture Modeling Framework

The Predictive, Probabilistic Architecture Modeling Framework (P²AMF)(Pontus Johnson, Ullberg, Buschle, Franke, & Khurram, 2013) is an extension of OCL (Object Management Group, 2010) for probabilistic assessment and prediction of system properties. The main feature of P²AMF is its ability to express uncertainties of objects, relations and attributes in UML-models and perform probabilistic assessments incorporating these uncertainties.

A typical usage of P²AMF would be to create a model for predicting, e.g., the availability of an application. In P²AMF, two kinds of uncertainty are introduced. First, attributes may be stochastic. When attributes are instantiated, their values are expressed as probability distributions (probability distributions are covered in section 5):

myServer.availability=normal(0.8, 0.1)

Second, the existence of objects and relationships may be uncertain. It may, be the case that one no longer knows whether a specific server is still in service. This is a case of object existence uncertainty. Such uncertainty is specified using an existence attribute *E* that is mandatory for all classes (here using the concept class in the regular object-oriented aspect of the word), where the probability distribution of the instance *myServer.E* might be:

$P(myServer.E)=0.8$

i.e. there is an 80% chance that *myServer still exists*. It might also be uncertain whether *myServer* is still serving a specific application, i.e. whether there is a connection between the server and the application. Similarly, relationship uncertainty is specified with an existence attribute *E* on the relationships.

The probabilistic aspects are considered in a Monte-Carlo fashion: For each iteration, the stochastic variables are instantiated with instance values according to their respective distribution. This includes the existence of classes and relationships, which are sometimes instantiated, sometimes not, depending on the distribution. Then, each of the P²AMF statements is transformed into a proper OCL statement and can be evaluated.

For more information on the theory of the P²AMF consider (Pontus Johnson et al., 2013). The used sampling algorithms are described in 4.2.3.

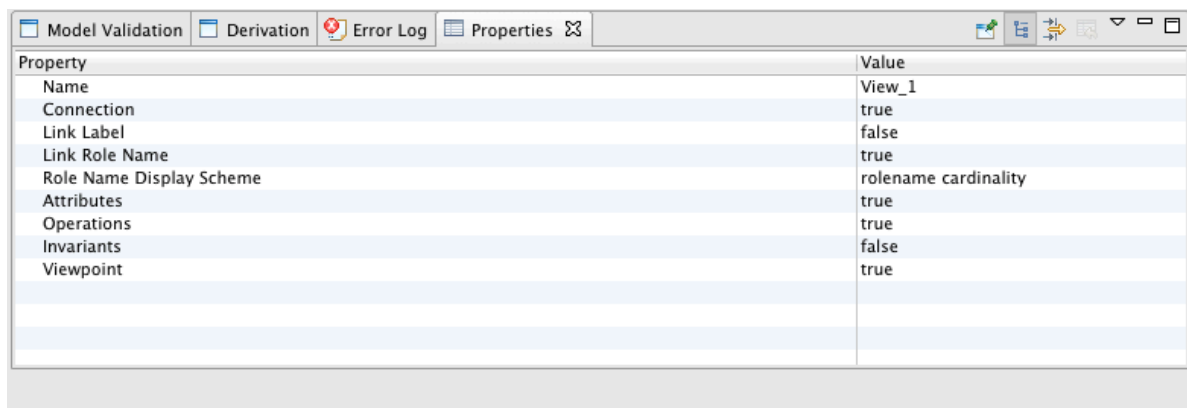
3 The Class modeler

3.1 General info

As mentioned in section 1, the final output of the class modeler is an assessment framework incorporated in a class model. In the following at first the overall user interface of the class modeler is described. Thereafter the creation of an assessment framework is explained. Finally additional tool features that not necessarily need to be used are described.

3.2 The user interface

The user interface of the class modeler (cf. Figure 6) mainly consists of the modeling canvas, which can be found in the center. This is the location where during the application of the class modeler the assessment framework will be created. To the right of the model canvas a palette can be found (cf. Figure 8). This palette holds all the possible model elements that can be used in order to construct an assessment framework. The user can hide the palette clicking the arrow located to the top right. Below the modeling canvas an area consisting off several tabs can be found (cf. Figure 2).



Property	Value
Name	View_1
Connection	true
Link Label	false
Link Role Name	true
Role Name Display Scheme	rolename cardinality
Attributes	true
Operations	true
Invariants	false
Viewpoint	true

Figure 2 Tabs to consider and specify model properties can be found in the lower part

These tabs show the outcome of model validation, allow specifying P2AMF based derivations, visualize the error log and allow investigating and changing properties of the model elements. In the lower left corner a model outline can be found (cf. Figure 3).

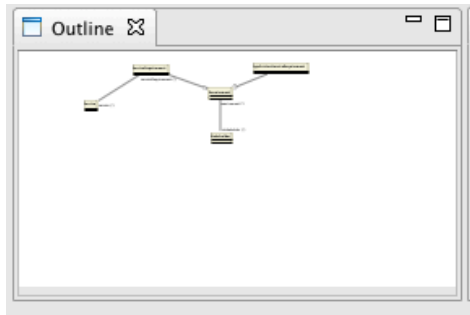


Figure 3 The model outline

This outline contains allows to quickly navigate through the model and consider it from a high level. To the left of the modeling canvas three tabs can be found (cf. Figure 4).

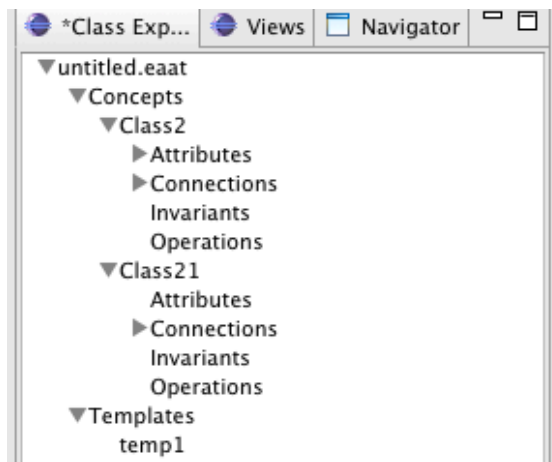


Figure 4 Tabs allowing navigating through the model

These tabs allow exploring the content of the created class model and its visual structure. One tab, the Class Explorer, lists all the created classes, their properties and the defined templates. A second tab follows, Views, allowing the consideration of different views on the model. The tab navigator allows investigating the currently considered view.

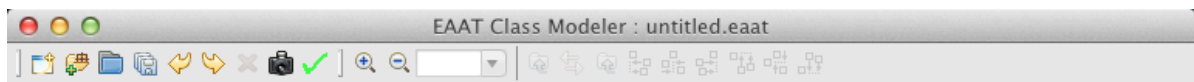


Figure 5 The menu bar

On top of the modeling canvas a menu bar can be found. This bar allows quick access to several frequently used functionalities. Both new Views and Templates can be defined based on classes of the model. New models can be opened and the existing one can be saved (including save as...). Thereafter the menu bar offers functionality to undo and redo the last performed activity. Further, the menu bar offers delete functionality,

export of the currently considered view/template to png and model validation. This functionality is followed by zooming functionality. Thereafter the user interface offers several alignments options. The menu of the tool is located in the top area of the user interface. Here, the File menu offers to save, load and export class models. The Edit menu covers undo/redo, delete, cut, copy & paste, and select all. In the P2AMF menu a refactoring of the P2AMF code is offered. A switch to turn on auto validation and an option to validate the model follow. Finally the Show menu allows to show all the dialogs and elements of the user interface again, if they have been closed.

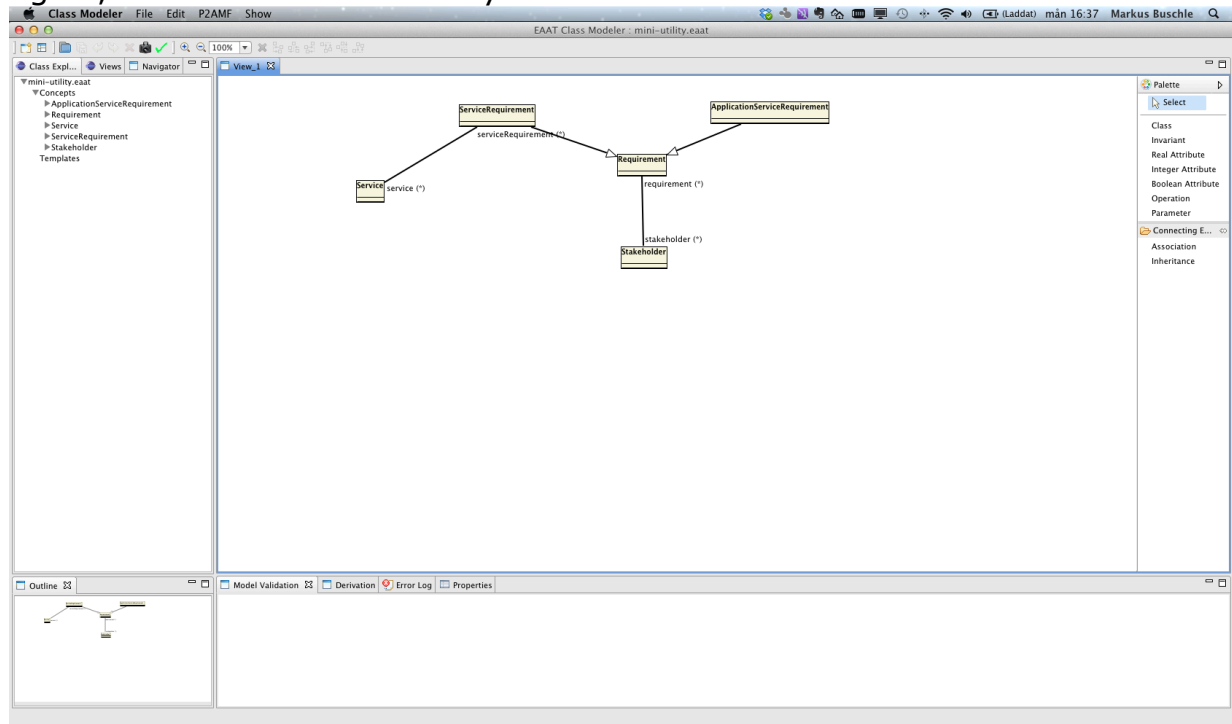


Figure 6 The user interface of the class modeler

3.3 Modeling process

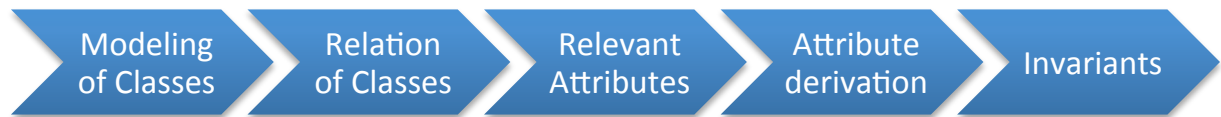


Figure 7 Recommended process for the creation of class models

The creation of assessment frameworks typically follows the process described in Figure 7. A user can perform the process steps in any order or divided over several iterations.

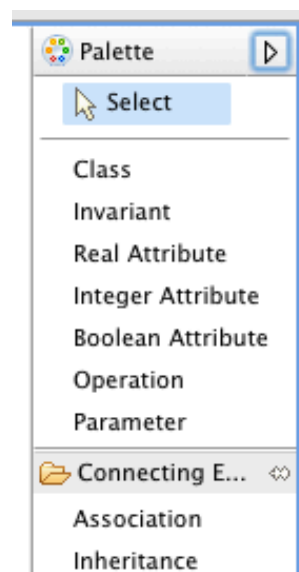
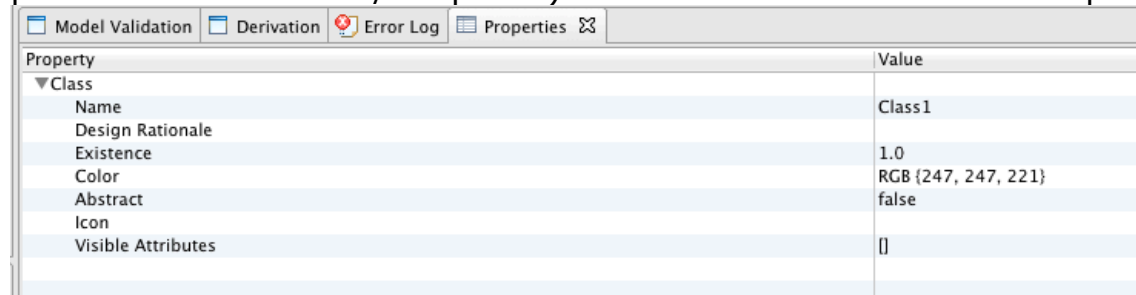


Figure 8 The palette

The following descriptions assume that the process is followed. The application of the class modeler starts by modeling the relevant classes. Classes can be added to a model from the palette (cf. Figure 8). At first one needs to select Class and then drag it out to the modeling canvas. Now the name of the class can be adjusted by double clicking the top part of the newly added class. Additionally it's possible to change the name from the properties tab (as described above, located in the lower part of the user interface). This tab (cf. Figure 9), once a class was selected, also offers to specify a design rational, i.e. a short notice on the modeled class. The existence, according to P²AMF can be specified and the background

color can be set. The class can be set to be abstract. It is further possible to assign an icon to that class. Finally the visible attributes (for that particular view/templates) can be specified.



Property	Value
▼ Class	
Name	Class1
Design Rationale	
Existence	1.0
Color	RGB {247, 247, 221}
Abstract	false
Icon	
Visible Attributes	[]

Figure 9 The properties tab for classes

If a class should describe a specialization of an already modeled class an inheritance relation can be used. From the palette the connecting element inheritance can be selected for this purpose. Starting from the specialization and then dragging a connection to the general class creates an inheritance relationship. This can for example be used to express that the class Linux is a specialization of the class Operating System. In a model one would an inheritance relation from Linux to Operating System.

Another way of connecting two classes is using the association relation. Associations can be created from the palette too. Once association was selected relations will be created between any two classes that are selected in order. A fast way to relate two classes is to select the first class, then press the alt key and drag a connection to the target class. Once the cursor is over the target one needs to drop the connection. The properties of the association relationship can be modified from the properties tab (cf. Figure 10). The name of the association can be changed. It is also possible to specify a design rationale and to set the existence of the relation according to P²AMF. Further multiplicity and roles can be set. Furthermore OCL properties of the relation can be changed. The relation can be set to be derived, a containment, unique and ordered. Additionally it's also possible to change from an association to an aggregation or composition relation. The derivation statement, in case of a derived relation, can be specified in the derivation tab. It is possible to specify a priority for a derived relation. This priority specifies when the derivation will take place (compared to the derivation of other derived relations). The tool uses 0 as the highest value and all values below (e.g. 1, 4, ...) as lower prioritizations. This can be helpful when relations should be derived considering other derived relations.

Model Validation Derivation Error Log Properties	
Property	Value
Label	Ref1
Design Rationale	
Existence	1.0
Priority	0
Class2 to Class21 Multiplicity	1
Class2 to Class21 Role Label	class21
Class21 to Class2 Multiplicity	1
Class21 to Class2 Role label	class2
Derived	true
Containment	true
Unique	true
Ordered	false
Type	Association

Figure 10 The properties tab for relations

One can use attributes to describe classes in the context of the assessment framework. The class modeler supports three different types: real, integer and Boolean. These three types can be found in the palette. Classes can be described with them by dragging an attribute from the palette to the class of interest. There are two types of attributes that can be used, derived and non-derived ones. In order to specify how an attribute should be derived based on other attributes and/or the structure of the model a derivation statement needs to be specified. This can be done from the derivation tab. Attributes can be modified from their properties tab. Their name can be changed, a design rationale can be provided, it can be set if the attribute should be derived and the type can be set. Furthermore, already in the class modeler, the visual appearance of the attribute in the object modeler after calculations can be defined. Therefore the tab item color option allows assigning colors for the calculation results. It is possible to assign colors to expected states (cf. Figure 11). The calculated attributes in the object modeler will then be colored based on their values, relatively to the defined states. A gradient shows the specified states and the colors that can be expected.

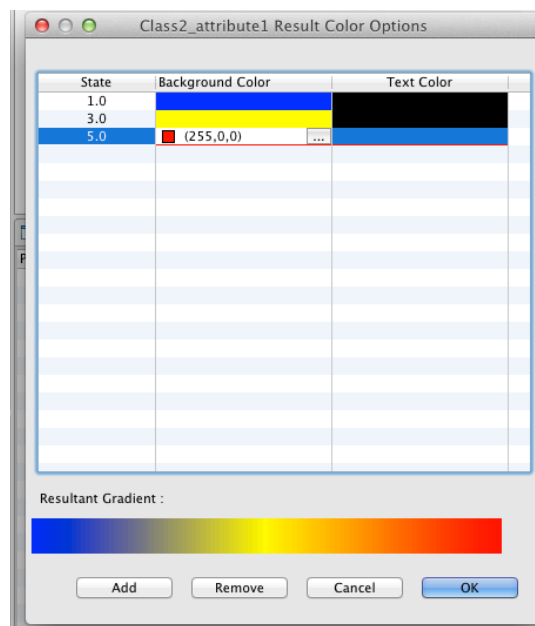


Figure 11 The dialog for assigning calculation results to visualization colors

Among others to foster code reuse P²AMF allows to define operations to be used during attribute derivation. To add operations to a class one needs to drag the operation element from the palette to the object that should be equipped with it. From the properties tab these operations can be customized. A new name can be set and a design rationale can be provided. Further the output type can be specified (yet again from real, integer and Boolean). It can also be specified if the result should be a single element or a collection.

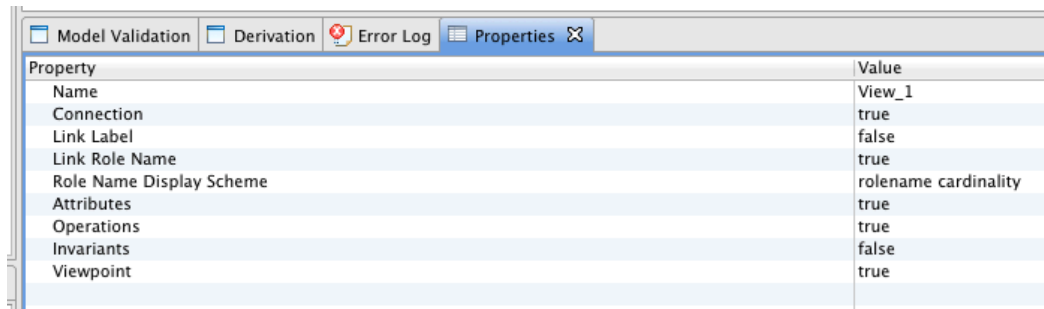
Furthermore parameters can be added describing the input of operations. Yet again, this is done from the palette. A new parameter can be dragged from the palette to the desired operation, where it then can be dropped. Once a parameter is dropped it can be adapted, based on the user's intention. From the properties tab it is possible to change the name, assign a design rationale, change the type (real, integer and Boolean) and set whether the parameter should describe a set (including bag and sequence) or a single element.

As described in Figure 7 it is possible to specify invariants on the model. Invariants are constraints on the model that need to hold during the creation of object models. Those can for example specify the values of attributes, number of instantiations, or structure of relations. Yet again, these can be found in the palette and can be assigned to elements of the model by dragging and dropping. Their evaluation can be specified in the derivation tab.

A click on the plain canvas allows to set general properties of the model (cf. Figure 12). Again, this can be done from the properties tab. This tab allows changing the name of the currently considered view. Furthermore it can be selected if connections, link labels and role names should be shown. The pattern to display role names can be adapted too. Additionally the visibility of attributes, operations and invariants can be specified. Finally it is possible to set this view as a viewpoint (to be used in the object modeler).

3.4 Reducing the visual complexity

The content of the modeling canvas gets typically harder to grasp after some time of modeling with more and more elements added to the modeling. The class modeler provides two means of abstraction in order to reduce the visual complexity.



Property	Value
Name	View_1
Connection	true
Link Label	false
Link Role Name	true
Role Name Display Scheme	rolename cardinality
Attributes	true
Operations	true
Invariants	false
Viewpoint	true

Figure 12 The properties tab for views

On the one hand templates can be created composed out of already defined classes. On the other hand views can be used to visually divide the model into smaller, easier to understand, subsets.



Figure 13 The new template button in the menu bar

To create a template one clicks the new template button from the menu bar (cf.

bar (cf.

Figure 13). Thereafter a new template is added to the templates that are listed in the class explorer. A template can consist of classes or other templates. The user can connect the elements of a template based on the connections specified in the class model. The characteristics of a template can be modified from the properties tab. Here the name can be changed, a design rationale can be provided and some statistic is presented. Furthermore one can specify the existence of the model elements that are part of the template. It is also possible to change the color of the template to visually highlight it. Yet again, this can be done using the associations option from the palette or via a drag& drop relation (pressing the alt-button). Additionally the external behavior of templates can be configured i.e. how a template connects to the rest of the model. This connectivity is realized via ports that can be added from the palette. The modeling of a port is a two-step process. At first a relationship from the class model needs to be selected that should be used to build the connection upon. As a second step one needs to decide where this connection should interact with the template. Therefore a connecting class within the template needs to be selected. Ports can be configured from the properties tab. It is possible to adjust the name and provide a design rationale. Furthermore

statistics on the usage of that port is provided too. The visualization of instantiated templates can be defined from the class modeler. In order to do so one needs to first right-click on the template in the class explorer. A context menu pops up and allows adding a color profile. This profile can be modified from the properties tab. The option attributes allows selecting the attributes that should be used to evaluate the colors of a template. This is done considering the colors of the attributes as they have been specified before. If an attribute should be considered for the coloring of a template than it's visibility should be set to Yes. The object modeler will blend the color based on the selected attributes.

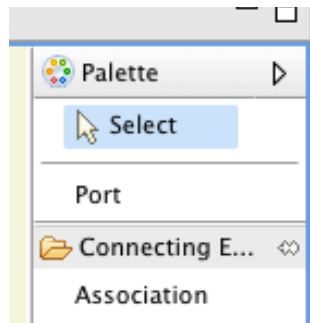


Figure 14 The palette for templates

It can be useful to restructure an existing model using views and break it down into subparts that are easier to understand. To add a new view based on the existing model clicking Add View from the menu bar is necessary. This adds a new empty canvas, which can be filled with either new classes or other new model elements or based on already modeled content. To add existing classes the user can drag them from the Class explorer or paste them (after copying) from an existing model. As described earlier: first clicking on the plain canvas and then navigating to the properties tab can change the properties of the view.

Once a model is complete or at any moment during the model creation the user can validate it. Validation here refers to checking if the model is a proper P²AMF model and can be used for calculations. This can be triggered from the P2AMF model. Results are displayed in the model validation tab, located in the lower part of the user interface. For validation, the class modeler translates the model into an OCL compatible model. Then the class modeler uses 3rd party libraries (provided by the eclipse modeling framework EMF) in order to check the syntax and semantics. The tool displays the results unchanged and in case of ambiguities it often helps to google the included key words.

4 The Object modeler

4.1 General info

EAAT Object modeler is an enterprise architecture modeling tool that uses UML notation for visual representation and a probabilistic calculation engine Probabilistic Architecture Modeling Framework (P2AMF) for analysis.

EAAT Object modeler is designed for modeling different properties of enterprise information systems like availability or interoperability. A property can be modeled once a framework (sometimes also called meta model) in a form of a class model for it exists. EAAT Class Modeler is the tool for creating the underlying class models for Object Modeler.

4.2 Modeling process

To create a new object model in Object Modeler one needs first a class model that describes the concepts and the theoretical background for the wanted model. The object models created in Object Modeler describe a real life situation in the language of the class model.

To create a model in Object Modeler one follows these steps:

- I. Understand the real life situation in terms of the concepts defined in the (EAAT Class Modeler) class model.
- II. Create the structure of the model in the tool by adding objects and drawing relationships between them.
- III. Map the data needs for object attributes (evidence) and gather/ or estimate it, then input it into the model.
- IV. Choose proper calculation method and the amount of samples used. Depending on the class model, size of the object model, and the precision required, about 1000 to 10 000+ samples might be appropriate.
- V. Possibly modify the model and redo steps from I to IV.
- VI. Interpret or compare the calculation results.

The modeling steps involving the tool are further explained in the following sections with screenshots. For the theory about enterprise architecture analysis please see (P. Johnson & Ekstedt, 2007).

4.2.1 Model structure

The visual modeling in EAAT Object Modeler utilizes the popular UML notation. A user has a set of concepts at his/her disposal, which are in the form of classes. A class is used to create (instantiate) objects of that type. Classes may contain attributes, operations and have restrictions associated with them. The exact nature of the classes, their properties and available connections depends on the underlying Class Modeler class model. Figure 15 shows the user interface of the Object Modeler tool. Here you can see four connected objects from an example class model. The focus of this class model is service availability.

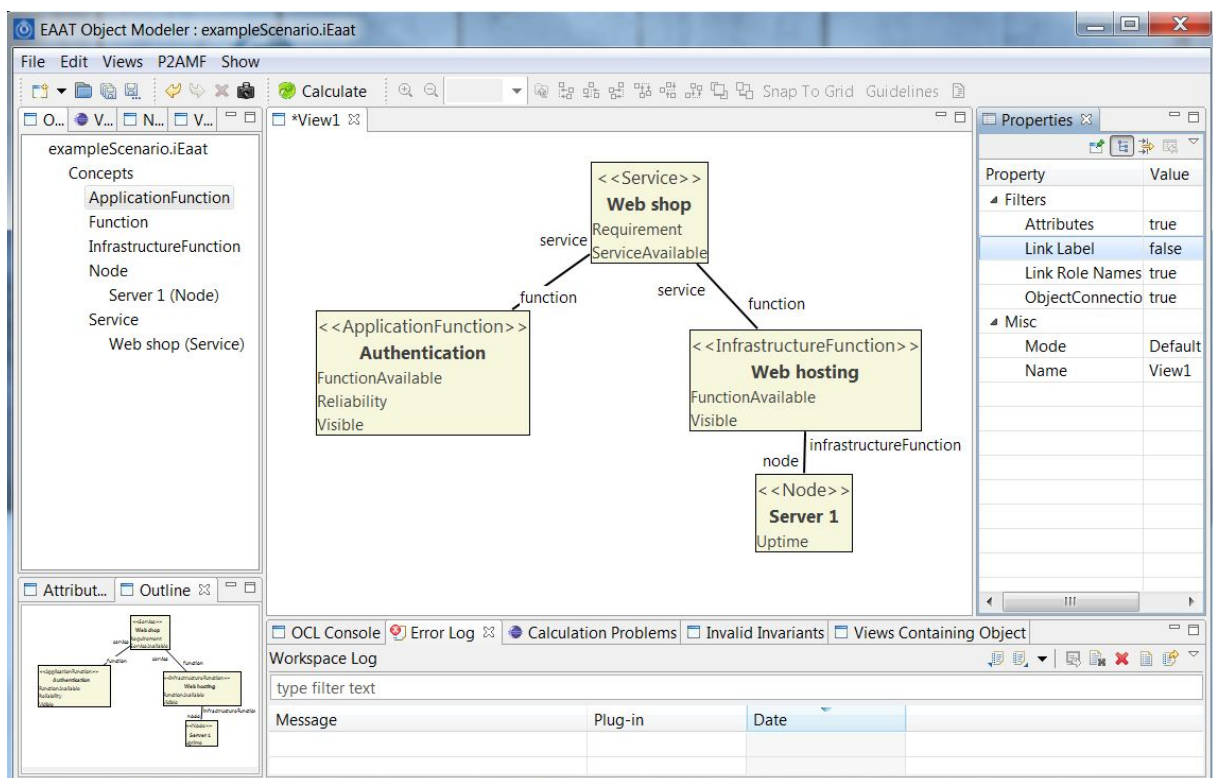


Figure 15 EAAT Object Modeler window with main canvas in center

The objects in Figure 15 are based on the classes of Service, ApplicationFunction, InfrastructureFunction and Node, and have been named according to a real life scenario Web shop, Authentication, Web hosting and Server 1. Naming elements in the model makes the model more comprehensible, so it is highly recommended.

On the left side of the main canvas is a window with several tabs that provide access to relevant areas for model creation and navigation. The first tab Object Model Explorer shows a list of all available concepts from the class model. These concepts can be dragged to the main canvas. By expanding the class (concept) name one can see how times the class has already been used in the model. Under each class name is a list of created

objects. In Figure 15 the classes of Node and Service have been expanded in the Object Model Explorer and the model contains one instance of each.

The Navigator tab takes us to a window that displays all created objects and their attributes. Clicking on an element in the window will center the corresponding object in the main canvas, and display corresponding attribute value in the Attribute Value window. Attribute Value window is located at the lower left corner together with the Model Outline window. The Attribute Value window shows values only after a calculation has taken place. If no element is selected, then the calculation time will be shown.

Besides the Object Model Explorer and Navigator windows, there are two other important windows that can be reached through tabs on the left. These are for views. View Explorer shows all the created views for the current Object Model. Viewpoint tab on the other hand shows all the views for the Class Modeler class model that the Object Modeler tool uses. Views can be changed in the tool, viewpoints not.

Every new model creation starts in "View 1", which is the default view. In addition to the default view, a user can create an arbitrary number of customized views. Customized views allow highlighting or separating different parts of the model for easier understanding and manipulation. The elements – objects and associations, can exist in the model without being visible in any of the views. However, they will be accessible from the window Object Model Explorer.

The properties of elements like objects, attributes, relationships and views are shown right of the main canvas. The properties of a view allow hiding certain elements. In figure 15 attributes, link role names, and object connections are displayed for View1, while link labels are hidden.

The new Object modeler supports templates. Templates are a way to group EAAT objects. These groups are defined in the Class Modeler tool and are part of the class model. Each of the defined groups has one or more connecting interfaces, which is how other objects are linked to them.

4.2.2 Evidence

The objects in Object Modeler usually have attributes. Some attributes obtain values during calculation process, while values for others are defined manually. The attributes with defined values are called non-derived attributes and the ones with generated values derived ones. The derived values can show important results for an analysis and are the reason why the object model was created in the first place. Several objects in Figure 15 have attributes, for example Web shop has attributes Requirement and ServiceAvailable.

Both types of attributes contain a property called Evidence that is the source of the attribute value. It is accessible from the attribute property sheet, called Properties. While the evidence for the derived attribute is

generated during the calculation process by a P2AMF statement, a non-derived attribute requires the user to set the value manually. If we don't set the evidence value, the field Statement will be used instead as evidence. This value has been set in the class model as a default value for this attribute. However, with certain types of calculation methods, evidence value can also be used for derived attributes. The details are explained in the section 4.3.

The attribute properties for Server 1 attribute Uptime are shown in Figure 16. Here we can see that the attribute value is not derived and should be entered. When it's not entered, a value from the Statement field will be used instead.

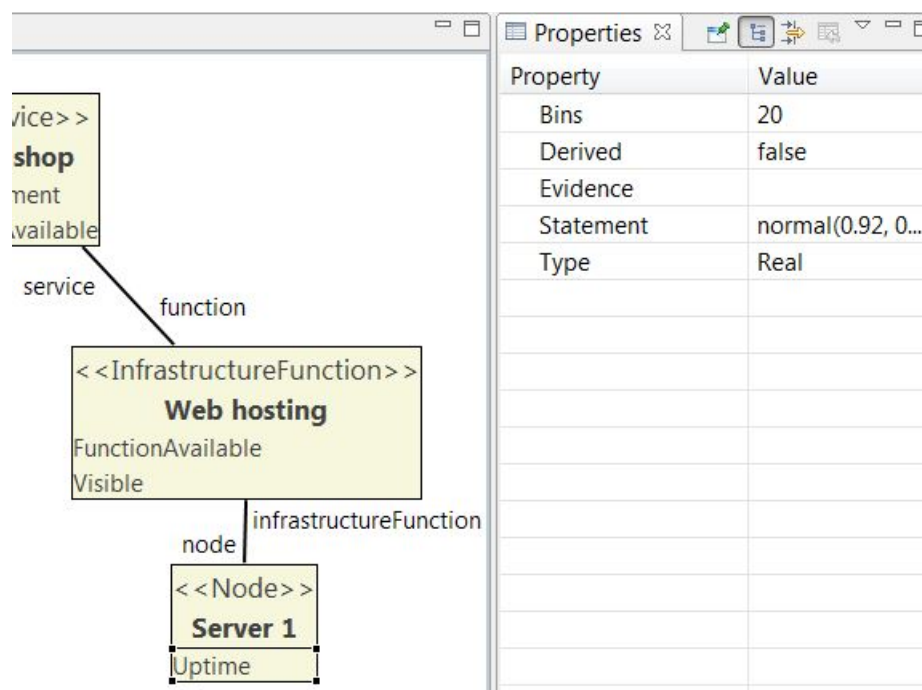


Figure 16 Properties of the Server 1 attribute Uptime

Another property that is visible from the attribute property sheet is the type of the evidence value. This value has been defined in the class model, and cannot be changed in the object model. It can be either integer, real number or Boolean.

The objects are connected with links, which show that there is a relationship between them. The links have role names on each end, which can be used by OCL code to navigate from one object to another and query the attribute values.

Object Modeler supports OCL statement execution. The statements can be either queries or code for debugging purpose, and cannot be saved. The statements can be launched from OCL console. The console is accessible from under the main canvas, under the tab OCL console. OCL statements are context based, which means that before any code can be executed from the console, a class has to be selected. All entered OCL statements

will then be evaluated in the context of the selected class. Figure 17 shows an example of evaluated code, where the code is entered into the lower half of the window, and the result is displayed in the upper half.

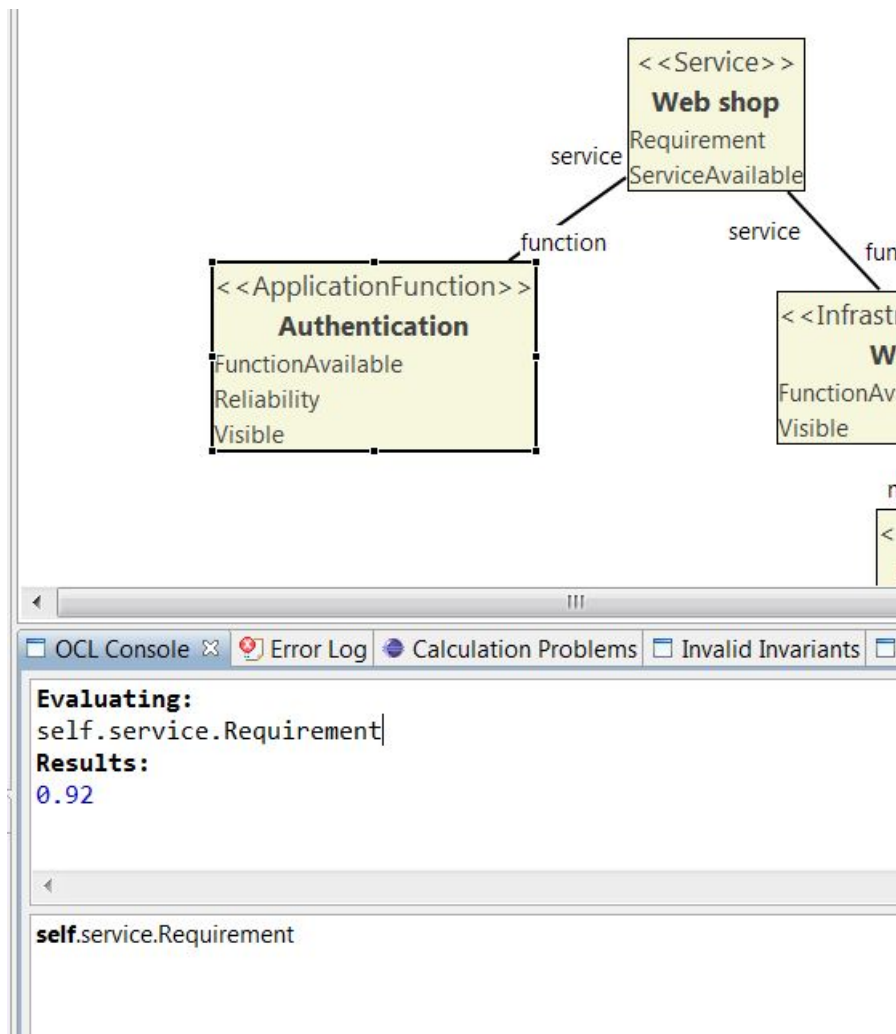


Figure 17 OCL Console in action

Here the class **Authentication** was selected and the following line of OCL code executed: *self.service.Requirement*. This code evaluated back to the value of the **Web shop** attribute **Requirement** value.

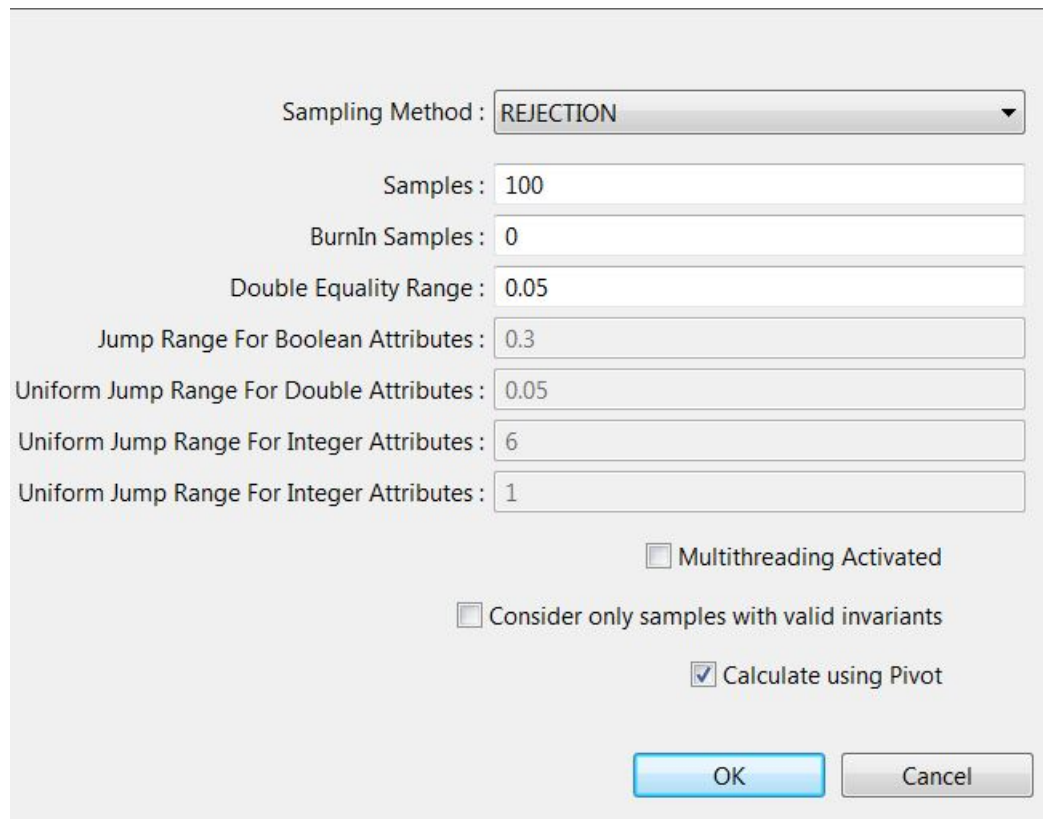
4.2.3 Calculation

This section describes how to launch a calculation, the calculation methods are discussed under a separate section 4.3.

The calculation in Object Modeler is done using a chosen sampling method. The sampling methods available are Forward, Rejection, Metropolis Hastings, Forward evidence injection sampling. Alternatively, the user can choose to do a deterministic calculation, where distributions are ignored and only deterministic parts of distribution functions are

extracted, to take only one sample. In case of a normal distribution that value would be the mean value.

Object Modeler allows to configure the sampling method and the number of samples. The configuration window can be accessed from the toolbar menu from under P2AMF > Configurations. The window is shown in Figure 18.



The image shows a configuration dialog box for P2AMF calculation. It contains several input fields and checkboxes. The 'Sampling Method' is set to 'REJECTION'. The 'Samples' field is 100, 'BurnIn Samples' is 0, 'Double Equality Range' is 0.05, 'Jump Range For Boolean Attributes' is 0.3, 'Uniform Jump Range For Double Attributes' is 0.05, 'Uniform Jump Range For Integer Attributes' is 6, and 'Uniform Jump Range For Integer Attributes' is 1. There are three checkboxes: 'Multithreading Activated' (unchecked), 'Consider only samples with valid invariants' (unchecked), and 'Calculate using Pivot' (checked). At the bottom are 'OK' and 'Cancel' buttons.

Parameter	Value
Sampling Method	REJECTION
Samples	100
BurnIn Samples	0
Double Equality Range	0.05
Jump Range For Boolean Attributes	0.3
Uniform Jump Range For Double Attributes	0.05
Uniform Jump Range For Integer Attributes	6
Uniform Jump Range For Integer Attributes	1

☐ Multithreading Activated

☐ Consider only samples with valid invariants

☒ Calculate using Pivot

OK Cancel

Figure 18 P2AMF calculation Configuration

The sampling methods can be changed from under the drop down menu Sampling Method. The number of samples needed depends on a sampling method.

Calculation is possible with both conventional Eclipse Modeling Framework and PIVOT OCL engines. Ticking the Pivot calculation off will revert to the conventional method. A user can choose only to do calculation with valid invariants. This will end in a lower calculation speed, but the results will correspond to the defined restrictions.

The calculation process can be launched from the toolbar or from menu, under P2AMF. Figure 19 shows the toolbar with Calculate button. Note that the green arrows next to the Calculate button are not part of it, but allow to change the underlying class model.

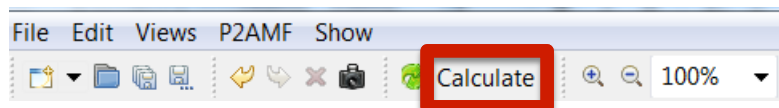


Figure 19 Toolbar with Calculate and other buttons

After calculation has taken place, results can be seen from the window on the left of the main canvas, called Attribute Value. After an attribute has been selected, the results are displayed in a form of a histogram. The value for Server 1 uptime is shown in Figure 20.

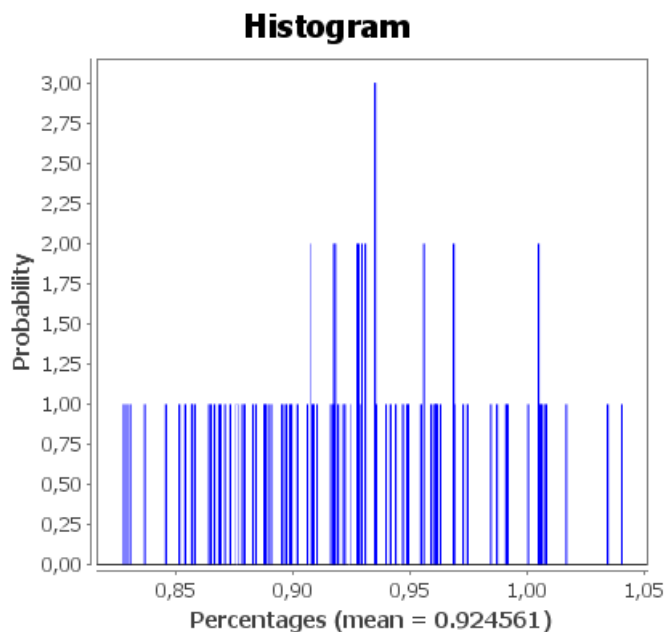


Figure 20 Attribute Value showing calculation results

Each bin (block) in Figure 20 represents a sample, and the Y axis shows the amount of samples that obtained the value that the X axis shows. For example Figure 20 shows that there were 3 samples that obtained the value of 0.935. The number of bins can be configured for each attribute separately. Default number is 20.

4.2.4 Exporting

An EAAT Object Modeler object model can be saved in a traditional way to a single file that contains both the Class and Object model. However, data can also be exported. The tool allows to export calculation results to Excel, images of the model and results, and the class model to a separate file.

The exported Excel file shows calculation results for all object attributes included in the model. The results are organized into columns that have titles "Object.attribute" and each row shows results for one sample calculation round.

There are two types of image exports supported. The open view export launched by the photo camera icon from the main toolbar, see Figure 19, and the attribute value export that can be launched from the down arrow in the Attribute Value window.

A right click on the Attribute Value and the choice of Properties will open the Chart Properties window. This window allows to configure the text and colors on the resulting image. See figure 21. The changes are useful when exporting calculation results.

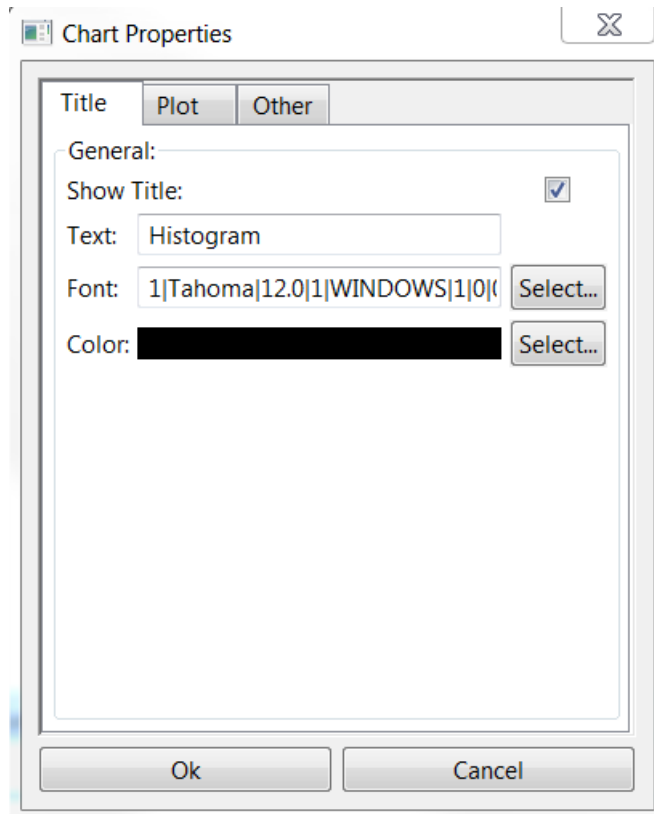


Figure 21 Attribute Value properties

4.3 Sampling methods

This section explains how inference is performed in the tool. The authors of P²AMF are not specific on how this should be realized. In the tool three sampling algorithms, to infer the values of the attributes that are part of the created model, are implemented: forward sampling (even in an extended version allowing to inject evidence), rejection sampling and Metropolis-Hastings sampling, each having advantages and disadvantages.

As described above, the user starts the sampling functionality as soon as the object model describes the scenario of interest. The P²AMF object model is sampled to create a set of deterministic object models. This is done considering the probability that a certain object is part of the created object model and that a given relationship is contained in that object model too. Therefore the existence attribute E , as explained above, is evaluated.

For each of these sample models, standard OCL inference is performed, thus generating sample values for all modeled attributes. For each attribute, the sample set collected from all sampled OCL models is used to characterize the posterior distribution.

For all sampling algorithms, the first step is to generate random samples from the existence attributes' probability distribution $P(X): x_1, \dots, x_M$. For each sample, x_i , and based on the P^2AMF object model O^p , a reduced object model, $N_i \in N$, containing only those objects and links whose existence attributes, X_j , were assigned the value true, is created. Some object models generated in this manner will not conform to the constraints of UML. In particular, object models may appear where a link is connected to only one or even zero objects. Such samples are rejected. Other generated object models will violate e.g. the multiplicity constraints of the class model. Such samples are also rejected. Additionally, some OCL derivations are undefined for certain object models, for instance a summation derivation over an empty set of attributes. A set of traditional UML/OCL object models remains with $\Xi \subset N$, whose structures vary but are syntactically correct, and whose attributes are not yet assigned values. Finally, if the user provides evidence for one or several attributes, the sample is assigned the evidence value.

4.3.1 Forward sampling

Forward sampling (cf. Algorithm 1) consists of only a few steps and leads to a fast sampling process. However forward sampling comes with the disadvantage of not allowing the specification of evidence; on any arbitrary attribute in the object models, only evidence on attributes not calculated based on other attributes' values is allowed. In the example depicted in () only evidence for the attribute *database server.availability* can be provided.

Forward sampling requires the attributes that are part of a sample Y_1, \dots, Y_n to be sorted in topological order i.e. parent attributes appear earlier in the sequence than the attributes that are calculated based on them, their children.

Following the general first step, as it was described above, the second step of the forward sampling algorithm is that for each of the remaining object models, Ξ_i , the probability distribution of the attributes not calculated based on the value(s) of other attributes, $P(Y^r)$ is sampled. This creates the sample set $y_1^r, \dots, y_{size(\Xi)}^r$. If there is evidence on a root attribute, the sample is assigned the evidence value. Based on the samples of the root attributes, the OCL derivations are calculated in topological order for each remaining attribute in the object model, $y_j^{\bar{r}} = f_{y_j^{\bar{r}}}(Pa_{y_j^{\bar{r}}})$. The result is a set of deterministic UML/OCL object models, $\Lambda \subset \Xi$, where in each model, all attributes are assigned values. The final set of object models, $O \subset \Lambda$, contains attribute samples from the posterior

probability distribution $P(X,Y|e)$. These samples may thus be used to approximate the posterior.

```

1   for (int i=1; i<M; i++) {
2       x=sampleExistenceAttributes();
3       N = extractObjectModel(Op, x);
4       if (syntacticallyCorrect(N)) {
5           for(int j=1; j<N; j++) {
6               Yj = sample(getParents(Yj));
7               Λ = assignAttributesToModel(y, N);
8               O.add(Λ);
9           }
10      }

```

Algorithm 1 Forward sampling

4.3.2 Forward sampling with evidence injection

Sometimes we know the value for derived attribute and we would like to consider only those samples that are in accordance with that value. To do that using forward sampling it is necessary to override the `assignAttributesToModel` function as it was presented in Algorithm 1. We replace the calculated value with our evidence and by doing so the calculations of the following children make use of the values provided by the user and not the derived ones. This leads to proper results, however modifies the structure of the model as it makes it partly inconsistent (for provided evidence children don't fit to their parents)

4.3.3 Rejection sampling

The objective of rejection sampling (cf. Algorithm 2) is to generate samples from the posterior probability distribution $P(X, Y|e)$, where $e = e^X \cup e^Y$ denotes the evidence of existence attributes as well as the remaining attributes. The objective is thus to approximate the probability distributions of all attributes, given that observations on the actual values of some attributes, and prior probability distributions representing beliefs about the values of all attributes prior to observing any evidence.

Rejection sampling extends the previously described forward sampling algorithm with a third step. In this third step object models containing attributes not conforming to the evidence are rejected.

The sampling process ensures that root attributes always do conform, but this is not the case for OCL- defined attributes.


```

1    for(int i=1; i<M; i++) {
2        x = sampleExistenceAttributes();
3        N = extractObjectModel(Op, x);
4        if (syntacticallyCorrect(N)) {
5            y = sampleRemainingAttributes();
6             $\Lambda$  = assignAttributesToModel(y, N);
7            if (conformsToEvidence( $\Lambda$ )) {
8                O.add( $\Lambda$ );
9            }
10       }
11   }

```

Algorithm 2 Rejection Sampling

As described above, rejection sampling extends forward sampling. Doing so, it overcomes the weakness of only allowing the specification of evidence on the root attributes. The pseudo code above shows that this is implemented as a filter, where samples confirming to the evidence are kept and all others are rejected. This proceeding is costly, as it requires the creation of many samples in order to generate a sufficient number of valid samples.

4.3.4 Metropolis- Hastings sampling

Metropolis-Hastings (cf. Algorithm 3) is an iterative sampling technique converging to a desired distribution limit. It aims at creating a Markov chain MC with a stationary distribution being the desired distribution, i.e., a chain of samples where the sampled attribute values match the specified evidence.

First one valid sample is created using rejection sampling. Once this sample is found it is used as the first element in the Markov chain.

```

1   $\Lambda_{init}$  = rejectionSampling(M = 1)
2  MC.add( $\Lambda_{init}$ )
3  for(int i=1; i<M+B; i++) {
4       $\Lambda$  = MC.getLast();
5       $P(\Lambda'|\Lambda)$  = calculateProbability( $\Lambda'$ ,  $\Lambda$ );
6       $\Lambda'$  = generateNewSample( $\Lambda$ );
7       $\alpha$  = calculateProbabilityOfAcceptance( $\Lambda'$ ,  $\Lambda$ ,  $P(\Lambda'|\Lambda)$ );
8      if ( $\alpha < 1$ ) {
9          O.add( $\Lambda'$ )
10     }
11     else {
12         O.add( $\Lambda$ )
13     }
14 }
15 removeBurn-InSamples(B)

```

Algorithm 3 Metropolis-Hastings sampling

The second step is to create a new chain element based on the last added element. A new sample is created as a copy of the last chain element. For the attributes without any specified evidence, new values are generated using a candidate-generating distribution. Then the likelihood of the new sample given the old sample $P(x'|x)$ is evaluated. Thereafter the probability of acceptance α of the sample is calculated, considering the likelihood $P(x'|x)$, which over time is given more consideration. If α is greater than a given limit l the sample is added to the chain otherwise the last added element is added again. The second step is repeated until a predefined number M of chain elements has been added.

The first samples are typically not used to evaluate the model; they are called burn-in samples B and train the algorithm. As a final step the burn-in samples are removed.

Similar to rejection sampling, Metropolis-Hastings sampling allows specifying evidence for any attribute of the model. This algorithm does need a comparably fewer number of samples and is therefore, especially when considering models including a large number of attributes, more effective. The biggest disadvantage of Metropolis-Hastings sampling is that, especially for models with many local minima, the best solution might not be found. This is because of the chain structure of the result, where samples are based on their predecessor.

4.4 Problems

This section describes how Object Modeler reports various problems and errors encountered during calculation process. Some of the more common errors are covered in FAQ.

Under the main canvas window one can find tabs for Error Log, Calculation Problems, Invalid Invariants. This is where any warnings or errors are reported. See Figure 22.

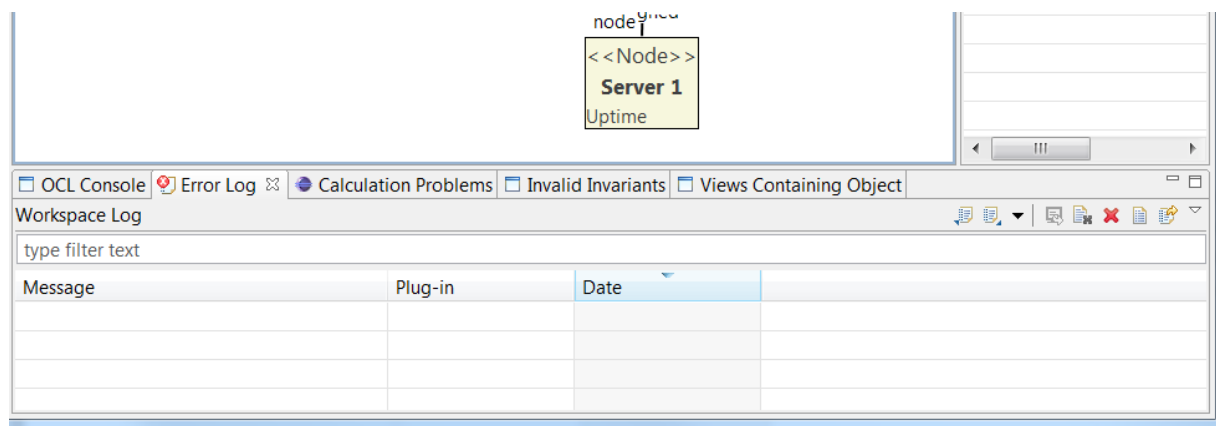


Figure 22 Warnings and errors

Error log displays errors and warnings together with the date and time when the event occurred. Each message can be opened to see event details and exception stack trace. This is useful for bug reporting.

Calculation Problem shows problems related to connection multiplicities. For example if A and B are two classes connected with each other and A to B multiplicities is 1..*. So it means instance of A must be connected to at least one instance of B. If this is not the case then calculation problems view indicates that.

Invalid invariants view show the invariants which are not validated on the current model. An invariant is a constraint on model/underlying system.

5 Specification of analysis

5.1 Mathematical functions

The following mathematical functions are available in EAAT. They can be used as part of code in EAAT Class Modeler, or as evidence in Object Modeler.

Note that, in current implementation, only probability distribution functions can be assigned to non-derived attributes, not cumulative distribution functions.

5.1.1 Probability Distribution Functions

Bernoulli distribution

Syntax: Bernoulli(double x) : boolean

- x denotes the success probability

Explanation: Bernoulli distribution is a discrete probability distribution, which takes value 1 with success probability and value 0 with failure probability. It is a special case of the binomial distribution with only 1 try.

Example: Bernoulli(0.8)

Normal distribution

Syntax: Normal(double x, double y) : double

- x is the mean value
- y represents the standard deviation
- Throws Exception - if sd \leq 0.

Explanation:

The normal, or Gaussian, distribution is a continuous probability distribution. A continuous probability distribution is a distribution that has a probability density function.

Example: Normal(100,0.3)

Lognormal distribution

Syntax: Lognormal(double x, double y) : double

- x is the scale value
- y represents the shape
- Throws Exception - if shape \leq 0

Explanation:

A log-normal distribution is a continuous probability distribution of a random variable whose logarithm is normally distributed. A variable might be modeled as log-normal if it can be thought of as the multiplicative product of many independent positive random variables.

Example: Lognormal(100,10)

Binomial distribution

Syntax: Binomial(int x, double y) : int

- x is the number of trials
- y probability of success
- Throws Exception - if trials < 0 or probability < 0 or probability > 1

Explanation:

The binomial distribution is the discrete probability distribution. It shows the number of successes in a sequence of independent yes/no trials.

Example: Binomial(100,0.34)

5.1.2 Cumulative Distribution Functions

Cumulative distribution function can be used as a parameter to a probability distribution function or to a cumulative distribution function.

Gamma function

Syntax: Gamma(double alpha, double beta, double x) : double

- Alpha is the scale parameter of this distribution
- Beta is the shape parameter for this distribution
- x is the point at which the function is evaluated.
- Throws Exception - if alpha <= 0 or beta <= 0

Explanation:

The gamma function creates a new gamma distribution with the given alpha and beta values. For a random variable X whose values are distributed according to this distribution, this method returns $P(X \leq x)$.

Example: Gamma(100,50,0.9)

Exponential distribution

Syntax: Exponential(double x) : double

- Mean x

Explanation: Creates exponential distribution with the given mean.

Example: Exp(25.7)

Pareto distribution

Syntax: `Pareto(double scale, double shape, double x) : double`

- Scale is the scale parameter of this distribution
- Shape is the shape parameter for this distribution
- x is the point at which the function is evaluated
- Throws Exception - if `scale <= 0` or `shape <= 0`

Explanation: Creates a Pareto distribution using the specified scale and shape.

Example: `Pareto(100,50,2.5)`

Linear Interpolation

Syntax: `Linear Interpolation(double[] s1, double[] s2, double w) : double`

- s1 is a set of double values
- s2 is a set of double values
- w is a double value.

Explanation:

Linear interpolation is a method of curve fitting with linear polynomials.

Example: `Linear([2,4,6,8],[1,3,5,7],3)`

5.1.3 Nested distributions

The basic purpose of nested distribution is to use the output of an inner distribution function as an input parameter for the outer distribution. Users can define distribution variables with nested distribution operations.

Not to be confused with the nested sampling algorithm from Bayesian statistics.

Example: *let xyz : Real = normal(100, exp(class1.attribute1, class2.test)) in*

In the given example, `class1.attribute1` and `class2.test` both are non derived attributes. The outer most distribution function will be used for sampling but inner distribution will be used to acquire cumulative Probability.

Note: attribute used inside probability functions must be non derived attributes. Also, attribute should not have any distribution function in its statement, meaning only attribute with constant values are allowed to be parameters. In case of multiple instance of attributes are available in `sample(multiple class instances)` then aggregated value(summation) will be used.

Example 2: *let xyz : Real = normal(100, linear([1,2,3], [0.2,0.3,0.4], class2.test)) in*

5.2 OCL

OCL is a standard of the Object Management Group (OMG) that adds quantitative expressive power to UML models. The information that OCL adds to the UML models can be broadly divided into two: constraints and queries. It can be used to describe suitable values for properties, pre and post conditions for operations, calculated property values, and object queries.

Every OCL expression must have context. Almost any element in the model can be declared as context, including objects (classes), attributes, operations and associations.

OCL is a strongly typed language. Some of the more common basic types available are: Boolean, Integer, Real, String, OclInvalid, OclVoid, UnlimitedNatural.

OCL supports collections, and the ones available are Bag, Sequence, Set, OrderedSet.

It is possible to use collections with parts that are of distinct type. These kind of collections are called tuples. Each part in tuple must have a name.

Tuple {a: Collection(Integer) = Set{1, 3, 4}, b: String = 'foo,' c: String = 'bar'}

Associations can be navigated to refer to other objects and properties. The navigation starts from a specific object on a class diagram by using the opposite navigation end, like *object.associationEndName*

Navigation to association classes can be done by using a dot and the name of the association class. In this case the expression evaluates to a subset. For example *self.Job* is an example of navigation to a class.

Navigation from association class to the objects always return exactly one object that can be used as a set by using an arrow (->). Navigating recursive association classes is possible by specifying the role name of the direction.

To avoid type conformance errors, it is desirable in certain circumstances to re-type the object. Like for example when the actual type of the object is the subtype. *oclAsType(Classifier)* can be used for that purpose. However, an object can be retyped only to a type to which it conforms.

Single navigation over an association results in a Set, combined navigations in a Bag, navigations with ordered results in an OrderedSet. The elements of a collection are separated by curly brackets and separated by commas. An example would be *Set { 1 , 2 , 5 , 88 }*

OCL supports several collection operations that allow building sophisticated queries, for example select, reject, and collect operations. While select and reject result in a sub collection of the original one, collect doesn't have that restriction.

A typical syntax of select, collect and reject operations:

collection->select(...)

collection->select(v | boolean-expression-with-v)

collection->select(v : Type | boolean-expression-with-v)

The resulting collection for collect in this case is not a Set but a Bag, that can have the same element more than once. It is possible to make a Set from the Bag by using asSet. For example, *self.employee->collect(birthDate)->asSet()*.

When an expression is used more than once it makes sense to define a sub expression. Let expression allows to do that. An example of let expression would be *let income : Integer = self.job.salary->sum() in*

Please see (Object Management Group, 2010) for more syntax and thorough explanations.

6 FAQ

6.1.1 Class Modeler

Question: I noticed that if I replace the class model then the names used in templates don't get updated in the object model. Is that by design?

Answer: It is by design. In meta model replacement, old names are kept so modeler can recognize the objects.

Question: Is it be possible to switch on relationship names in template view? I could not find it in the properties window.

Answer: In template view, user cannot switch relationship names. Template view represents instantiations of classes, so here it is not allowed.

6.1.2 Object Modeler

Question: I get problematic attribute error

Answer: Problematic attribute problem can have two reasons.

1. Error evaluating attribute
2. Evidence to derived attribute is conforming to sample value(only happens in Reject sampling, Forward sampling can solve this problem. P2AMF menu-> Configurations).

Question: I can see the following problem in Error Log: calculation result is empty, and attribute does not conform to the samples. What to do.

Answer: You might have an evidence for any attribute which does not conform with attribute value.

Either remove the evidence or use forward sampling for calculations.

Question: Can I use cumulative distributions as root distributions?

Answer: Cumulative distributions are not allowed to be used as root distributions.

Normal(100, any cumulative distribution(y,y)) is valid syntax so value for this distribution will be 100.

7 References

- Buschle, M., Johnson, P., & Shahzad, K. The Enterprise Architecture Analysis Tool – Support for the Predictive, Probabilistic Architecture Modeling Framework. , AMCIS 2013 Proceedings (2013). Retrieved from <http://aisel.aisnet.org/amcis2013/EnterpriseSystems/GeneralPresentations/15>
- Johnson, P., & Ekstedt, M. (2007). *Enterprise architecture: models and analyses for information systems decision making*. Studentlitteratur. Retrieved from http://scholar.google.se/scholar?hl=sv&q=Enterprise+Architecture+Models+and+Analyses+for+Information+Systems+Decision+Making&btnG=Search&as_ylo=&as_vis=0#0
- Johnson, Pontus, Ullberg, J., Buschle, M., Franke, U., & Khurram, S. (2013). P 2 AMF : Predictive , Probabilistic Architecture Modeling Framework. In *Enterprise Interoperability - Proceedings of the Fifth International IFIP Working Conference, IWEI 2013*.
- Object Management Group. (2010). *Object Constraint Language, Version 2.2*. Retrieved from <http://www.omg.org/spec/OCL/2.2>